SW Engineering CSC648 Summer 2021
DormMates

Milestone 4 • Version 1 • 30 July 2021

Team 01

Team Lead and Github master	Andrei Georgescu	ageorgescu@mail.sfsu.edu
Frontend Lead	Meeka Cayabyab	
Backend & Database Lead	Jonathan McGrath	
Engineer	Alexandre Ruffo	
Engineer	Jimmy Yeung	
Engineer	Sayed Hamid	
Engineer	Ahad Zafar	

History Table

Version	Date	Notes
M4V2	N/A	N/A
M4V1	07/30/2021	Initial submission
M3V2	07/30/2021	Addressed feedback
M3V1	07/22/2021	Initial submission
M2V2	07/19/2021	Addressed the vertical prototype, functional requirements, and diagrams feedback.
M2V1	07/08/2021	Initial submission
M1V2	07/01/2021	Added database master role to the title page, addressed use cases feedback, addressed functional requirements feedback, and addressed competitive analysis feedback.
M1V1	06/22/2021	Initial submission

Table of Contents

Product Summary

DormMates

Final Priority 1 Functions

1.1. <u>Unregistered User</u>
- 1.1.1. An unregistered user can create a new student or landlord account.
- 1.1.2. An unregistered user can view listings near an institution of their choice.
- 1.1.3. An unregistered user can view the Home page.
- 1.1.4. An unregistered user can view the Terms of Service page.
- 1.1.5. An unregistered user can view the FAQ page.
- 1.1.6. An unregistered user can view the Features pages.
- 1.1.7. An unregistered user can view the About page.

1.2. <u>Registered User</u>
- 1.2.1. A registered user should be able to login to with their username and password.
- 1.2.2. A registered user should be able to logout of their account.
- 1.2.3. A registered user should be able to change their username.
- 1.2.4. A registered user should be able to change their email address.
- 1.2.5. A registered user should be able to change their password.

1.3. <u>Students</u>
- 1.3.1. A student user must have a verified edu email.
- 1.3.2. A student user must have a completed profile.
- 1.3.3. A student user shall be able to view the student dashboard.
- 1.3.4. A student user shall be able to view student user profiles.
- 1.3.5. A student user shall be able to search for listings.
- 1.3.6. A student user shall be able to edit their personality.
- 1.3.7. A student user should be able to edit their schedule.
- 1.3.8. A student user should be able to edit their hobbies.
- 1.3.9. A student user shall be able to filter listings by amenities.
- 1.3.10. A student user shall be able to filter listings by distance from university.
- 1.3.11. A student user shall be able to filter roommate selections by personality.
- 1.3.12. A student user shall be able to filter roommate selections by major.
- 1.3.13. A student user shall be able to filter roommate selections by hobbies.
- 1.3.14. A student user shall be able to filter roommate selections by schedule.
- 1.3.15. A student user should be able to view the location of a listing on a map.
- 1.3.16. A student user shall be able to favorite a listing.

1.4. Landlords
   1.4.1. A landlord user shall be able to view the landlord dashboard.
   1.4.2. A landlord user shall be able to view their own profile.
   1.4.3. A landlord user shall be able to post listings.
   1.4.4. A landlord user shall be able to edit their listings.
   1.4.5. A landlord user shall be able to view student profiles.
   1.4.6. A landlord user shall be able to delete their listings.

Unique Features

DormMates targets college students who would like to search for both housing and roommates. Students who sign up on our website will have the ability to look for potential roommates who share the same interests using our roommate filtering system. Students will additionally have the option to filter through housing listings to find their ideal housing. Those who want to list housing can create a landlord account and post a listing as well.

URL

https://dormmates.net

Usability Test Plan

Test Objective: Create a Listing

This test is to create a listing. The test asks users to add fields such as amenities, description, and a price to a listing that will be stored in the database. This is being tested as it is a core feature of the platform that landlord users should be able to perform.

Test Description:

The system setup requires that the user is already a landlord user and they will begin on their dashboard page. Users should be on the URL https://www.dormmates.net/dashboard to begin testing and will measure if the process is simple and easy to perform.

Usability Task Description:

TASK	**DESCRIPTION**
Task	Adding amenities to a listing
Machine State	Listing is not created
Successful Completion Criteria	Listing has proper amount of amenities
Benchmark	Completed in 1 minute

TASK	**DESCRIPTION**
Task	Adding a description to a listing
Machine State	Listing is not created.
Successful Completion Criteria	Listing has description
Benchmark	Completed in 1 minute

TASK	**DESCRIPTION**
Task	Adding a price to a listing
Machine State	Listing his not created
Successful Completion Criteria	Listing has a proper price
Benchmark	Completed in 1 minute

<u>Test Objectives: Edit a Listing</u>

This test is to allow a landlord to edit an already created listing. The test will ask users to update fields that they wish to change such as the amenities, price, or description. This is being tested because it's a major functionality that a landlord user should be able to perform.

Test Description:

The system startup requires that the user already has a landlord account and a listing created. A user will begin testing on the URL https://www.dormmates.net/dashboard and they will measure for how simple and easy it was to update and if it properly updated the listing.

TASK	DESCRIPTION
Task	Edit amenities of a listing
Machine State	A listing has already a specified amount of amenities
Successful Completion Criteria	Added and/or removed the proper amount of amenities that the user has updated.
Benchmark	Completed in 1 minute

TASK	DESCRIPTION
Task	Edit price of a listing
Machine State	A listing has already a specified price
Successful Completion Criteria	Changed the current price to a new price
Benchmark	Completed in 1 minute

TASK	DESCRIPTION
Task	Edit description of a listing
Machine State	A listing has already a set description
Successful Completion Criteria	Changed the current description to a new description
Benchmark	Completed in 1 minute

Test Objective: Listing Favorites

This test is for the Favoriting function and its different uses. This is being tested because it's functionality for the student and also its purpose in displaying information about listings that students have favorited.

Test Description:

The system set up requires that the user is already a Student User. It also requires that the student be on a listing page or in order to view the favorites, on a student dashboard. The user should be on the URL https://www.dormmates.net/dashboard in order to start the testing. Intended users for the test are Student users who wish to favorite a listing for later viewing and for students who are on the dashboard and want to view their favorite listings on the student dashboard.

Usability Task Description:

TASK	**DESCRIPTION**
Task	Favoriting a Listing
Machine State	The listing is not favorited
Successful completion criteria	The listing has been favorited
Benchmark	Completed in a minute

TASK	**DESCRIPTION**
Task	View all created favorites
Machine State	No favorites are being shown
Successful completion criteria	All favorites of a user have been retrieved and displayed
Benchmark	Completed in a minute

TASK	**DESCRIPTION**
Task	Remove a Favorite Listing
Machine State	The listing has been favorited
Successful completion criteria	The listing is no longer favorited
Benchmark	Completed in a minute

<u>Test Objective: Listing Search</u>

This test is for the functionality of searching for a listing. We are testing the functionality of the search and all of it's result data also by query. This is tested due to it's imperative functionality for the student user and their ability to find nearby listings.

Test Description:

The system set up for this test requires the user to be a Student User and logged into the system. Also the user must be on the student dashboard. The starting point for the test is on the Student Dashboard. On the dashboard the user will navigate to the Search Listings page, and begin the test on the system.

Intended users for this test are users who are Student Users on the system and who are attempting to search for listings with different filters in their area. The user will begin the testing on the URL https://www.dormmates.net/dashboard.

Test Description:

TASK	**DESCRIPTION**
Task	Search for a listing
Machine State	No filters are being used
Successful completion criteria	Displays all listings that have a washer
Benchmark	Completed in 1 minute

TASK	**DESCRIPTION**
Task	Initiating a search for a listing by dryer filter
Machine State	No filters are being used
Successful completion criteria	Displays all listings that have a dryer
Benchmark	Completed in 1 minute

TASK	**DESCRIPTION**
Task	Initiating a search for a listing by wifi filter
Machine State	No filters are being used
Successful completion criteria	Displays all listings that have wifi
Benchmark	Completed in 1 minute

Test Objective: Roommate Search

This test is for the functionality of Searching for Roommates by Filters. We are testing the functionality of the Search and all of it’s result data also by query. This is tested due to it’s imperative functionality for the student user.

Test Description:

The system set up for this test requires the user to be a Student User and logged into the system. Also the user must be on the student dashboard.The starting point for the test is on the Student Dashboard. On the dashboard the user will navigate to the Search Roommates page, and begin the test on the system.

Intended users for this test are users who are Student Users on the system and who are attempting to search for students with different filters in their area. The user will begin the testing on the URL https://www.dormmates.net/dashboard.

TASK	**DESCRIPTION**
Task	Searching for Roommates
Machine State	No filters are being applied
Successful Completion Criteria	Displays all students with the proper personality
Benchmark	Completed in 1 minute

TASK	**DESCRIPTION**
Task	Searching for Roommates with a major filter
Machine State	No filters are being applied
Successful Completion Criteria	Displays all students with the proper major
Benchmark	Completed in 1 minute

TASK	**DESCRIPTION**
Task	Searching for Roommates with a schedule filter
Machine State	No filters are being applied
Successful Completion Criteria	Displays all students with the proper schedule
Benchmark	Completed in 1 minute

Usability Test Table:

Test/Use Case	% Completed	Errors	Comments	Average Time Spent
Listing Creation	100%	Unable to edit a listing Picture of my listing is not displaying	No message is displayed if I have no listings posted as a landlord user	2 min average Over expected time window
Favorites	100%		Does not notify me if it favorited the listing No message is displayed if I have no favorites No message is displayed when favorites are deleted	20 seconds average Within expected time window
Listing Search	90%		Limited filtering options Map display icons overlapping	
Listing Editing	100%	Picture is not being displayed after editing	Forces me to reupload a picture even when I don't want to change it.	
Roommate Search	100%		Limited Filtering options User cards do not display all of the filter options	

<u>Questionnaire</u>
Average of results of all testers

Questions	Strongly Disagree **1**	Disagree **2**	Neutral **3**	Agree **4**	Strongly Agree **5**
It was easy to edit a listing				X	
I found the systems too complex		X			
I found updating a description on a listing was easy to do					X
The fields were clear and concise with information needed to be entered					X
I was prompted with errors if I filled the fields in incorrectly				X	
Submitting each form was clear and easy				X	
It was easy and clear to favorite a listing				X	
It was easy and clear to view my favorites.					X
It was easy and clear to delete a favorite.					

Beginning each process was clear and intuitive					X
I found each process too complex.	X				
The filter options accurately filtered results					X
Results were easy to locate				X	
The results clearly show my filter options					X
The overall usability of this interface was excellent				X	

QA Test Plan

Test Objective

We will be testing the accuracy and stability of data to confirm that the site will be operating smoothly. Our unique features are based upon whether a user has registered an account as a student or a landlord therefore should be a prime focus on ensuring the quality of this process.

HW and SW setup

In order to access our website, users must have a working computer (Windows/MacOS) or a mobile device (iOS/Android) that has internet access. Users are able to access the website through browsers which includes Chrome, Safari, Microsoft Edge, and Mozilla Firefox. If a user signs up as a student account, the user must have a proper Edu email given to them by their institution. Once the credentials are met, the site can be reached by inputting https://dormmates.net/ in an eligible browser of their choice.

Feature to be Tested

1. The landlord dashboard page must only be available to landlord users.
2. The student dashboard page must only be available to verified student users.
3. All sensitive information must be encrypted before stored in the database.
4. Store users profile data on the database.
5. Store landlords listings on the database.

#	Description	Test Input	Expected Output	Pass/Fail
1	Landlord dashboard must only be available to landlord users.	Logged in student users will click on dashboard.	Lead the student user to the student dashboard page.	Pass
2	Student dashboard must only be available to student users.	Logged in landlord users will click on dashboard.	Lead the landlord user to the landlord dashboard page.	Pass
3	Unregistered users cannot access the dashboard.	dormmates.net/dashboard	Dashboard will not be shown in the navigation bar.	Pass
4	Attempting to create a Student user without an edu email.	User inputs a personal email, TestStudent@gmail.com	“University is not supported”	Pass
5	User creates password to be stored into the database as a hash	Password : “Jwalters34”	“Successfully created account” also a Hashed password in the database	Pass
6	User creates an account and types in a password.	Password : “Jwalters34”	Password will be hidden as it is typed in on the register page	Pass

7	Stores users ID as 36bit UUID	Finish Registration Form with correct fields	“Successfully created account” User Id will be encrypted hex in Database	Pass

8	Store users profile data on the database.	INSERT INTO
		
		user(id,username,emailAddress,password,name,birthdate,gender, photo, lastSeen)
		
		VALUES(UUID_TO_BIN(?, true),”Tommy”,”Tommy10@gmail.com”,PassW0rd21,”Tom Holland”,9-10-1990,Male, Photo,NOW())
			“Successfully created account”	Pass

9	Store Students profile data on the database	INSERT INTO
		
		student(id,userId,institutionID,major,personality,schedule,hobby1,hobby2)
		
		VALUES(UUID_TO_BIN(*, true),UUID_TO_BIN(*, true),economics,mediator, afternoon,dancing,gardening)
			“Student created”
			Along with a student object with the parameters passed
				Pass

10	Store landlords profile data on the database	INSERT INTO
		
		landlord(id,userId,rating,numOfRatings)
		
		VALUES(UUID_TO_BIN(*, true),UUID_TO_BIN(*, true), 0, 0)
			“Landlord created”
			Along with a landlord object
				Pass

11	Store Landlords listing data on the database	INSERT INTO
		
		listing(id,landlordId,price,location,description,verification,availability,listingLatitude,listingLongitude)
		
		VALUES(UUID_TO_BIN(*, true),UUID_TO_BIN(*, true),1000,description,true ,true,33,-121)
			“Listing Created”
			Along with a listings object with the parameters passed
				Pass

12	Store Listings Amenities on the database	INSERT INTO
		
		amenities(listingId,washer,dryer,wifi,closet,furnished,
		
		kitchen,whiteboard,bath,livingroom,patio,parking)
		
		VALUES(UUID_TO_BIN(*,true),0,1,1,1,1,0,1,1,1,true,false)
			“Amenities Created” and “Listing Created”
			Along with the parameters that were passed
				Pass

13	Remove Listing from database from landlord	DELETE FROM listing
		WHERE id =
		UUID_TO_BIN(*,true)
			“Listing deleted”	Fail

14	Delete listings Amenities from database	DELETE FROM listing
		WHERE id =
		UUID_TO_BIN(*,true)
			“Listing deleted”	Fail

15	Viewing favorite listings	Click on student Dashboard	Listings displayed	Pass

Code Reviews

Coding Style:

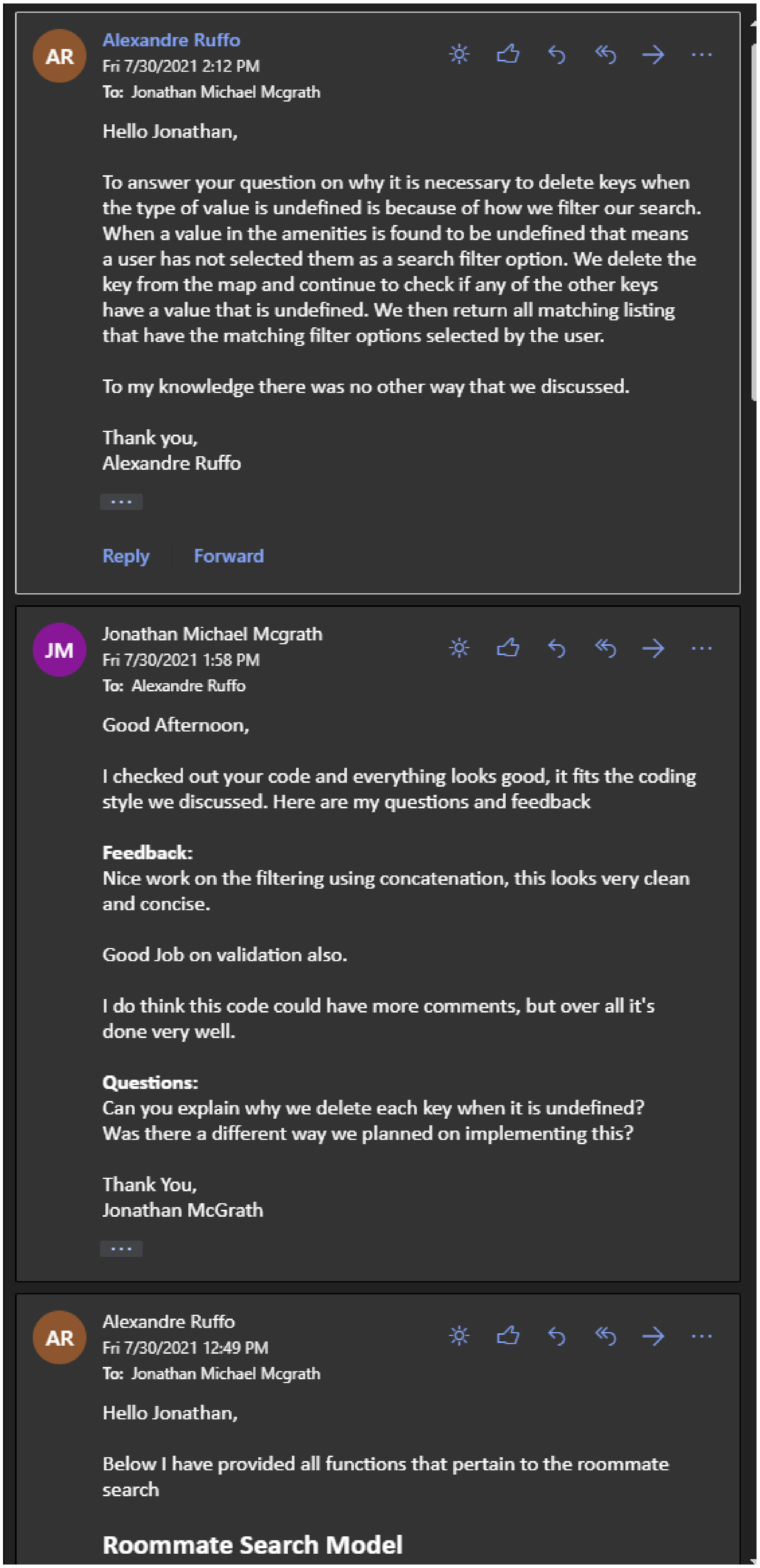
- File names will be universal with their naming conventions and entities
- Code will be organized and labeled appropriately with comments
- Headers will be clear and include file names and descriptions

BackEnd:

- Backend code will use camel case naming convention
- Helper functions will be notated and implemented where needed
- Naming convention will be universal through every route to model
- Parameters will match the naming convention of the database entities and attributes

Front End:

- Button tags will have matching names that follows the backend
- Div id will be unique so that they are to be used in javascript accordingly



Alexandre Ruffo
Fri 7/30/2021 2:12 PM
To: Jonathan Michael Mcgrath

Hello Jonathan,

To answer your question on why it is necessary to delete keys when the type of value is undefined is because of how we filter our search. When a value in the amenities is found to be undefined that means a user has not selected them as a search filter option. We delete the key from the map and continue to check if any of the other keys have a value that is undefined. We then return all matching listing that have the matching filter options selected by the user.

To my knowledge there was no other way that we discussed.

Thank you,
Alexandre Ruffo

...

Reply | Forward

Jonathan Michael Mcgrath
Fri 7/30/2021 1:58 PM
To: Alexandre Ruffo

Good Afternoon,

I checked out your code and everything looks good, it fits the coding style we discussed. Here are my questions and feedback

Feedback:
Nice work on the filtering using concatenation, this looks very clean and concise.

Good Job on validation also.

I do think this code could have more comments, but over all it's done very well.

Questions:
Can you explain why we delete each key when it is undefined?
Was there a different way we planned on implementing this?

Thank You,
Jonathan McGrath

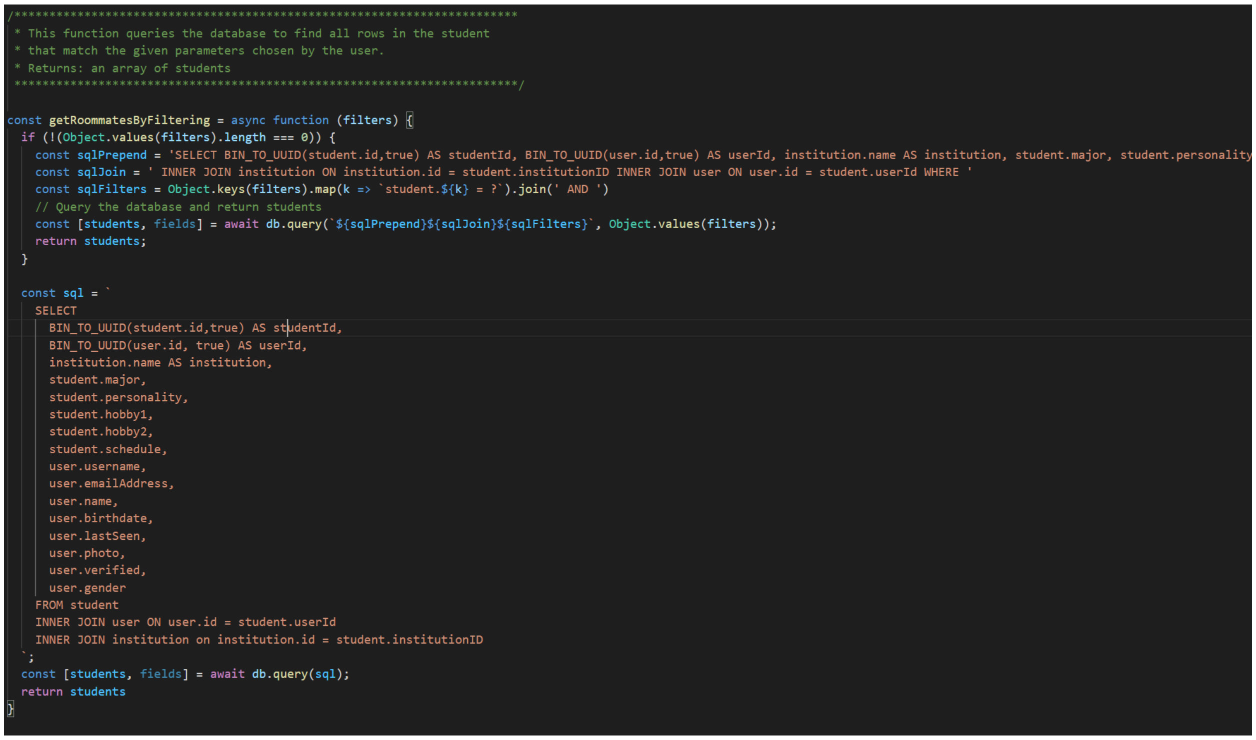
...

Alexandre Ruffo
Fri 7/30/2021 12:49 PM
To: Jonathan Michael Mcgrath

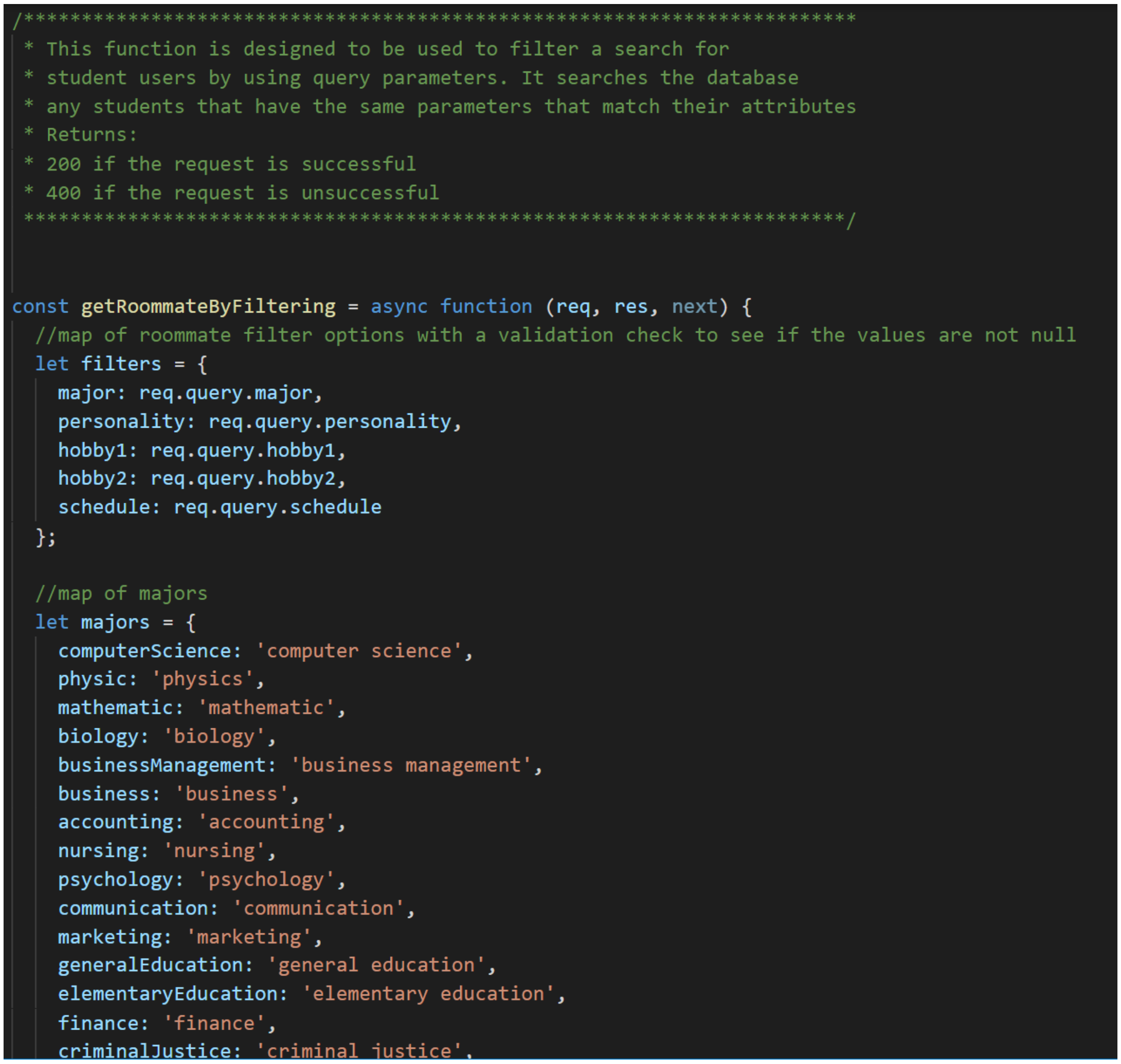
Hello Jonathan,

Below I have provided all functions that pertain to the roommate search

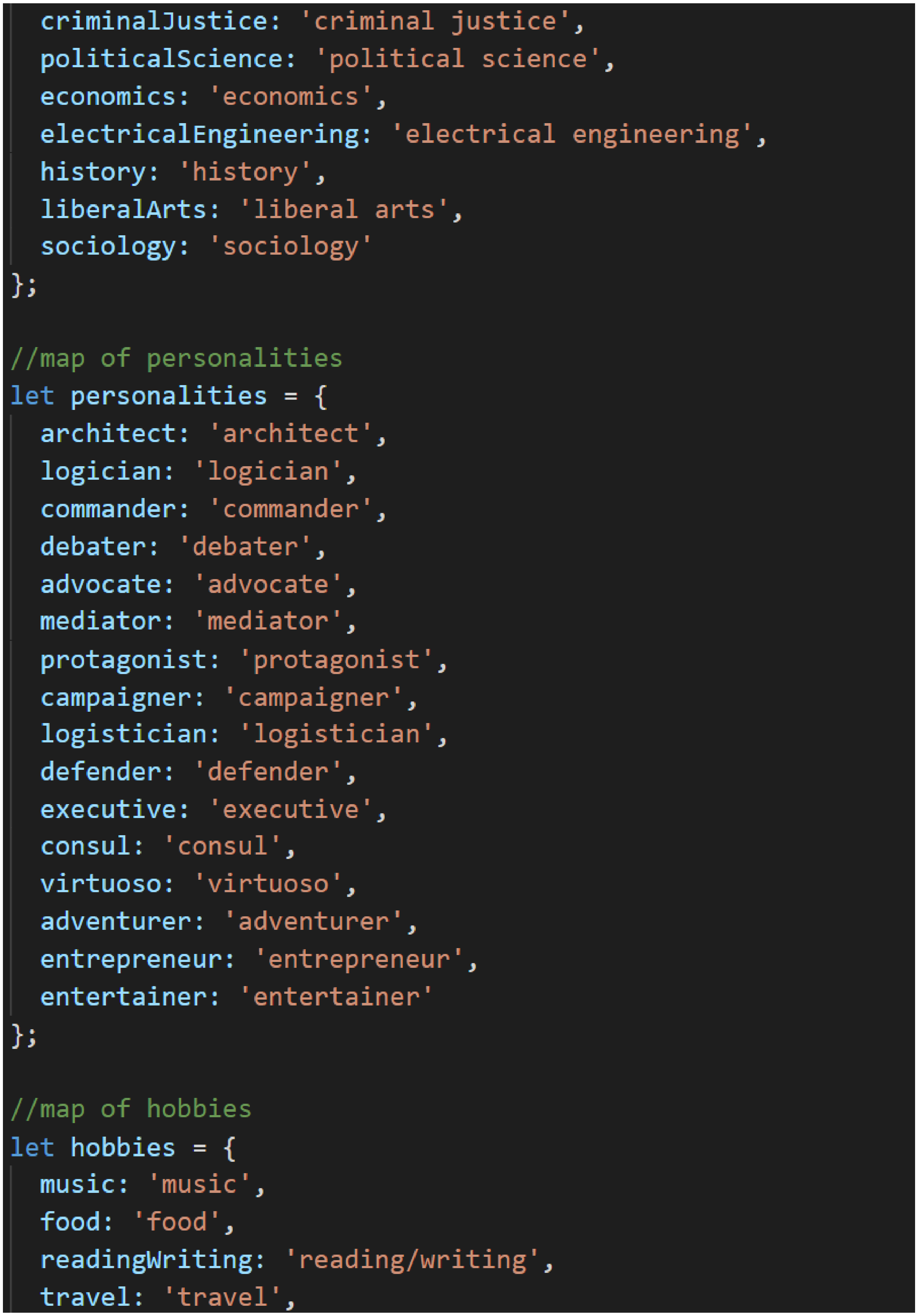
Roommate Search Model



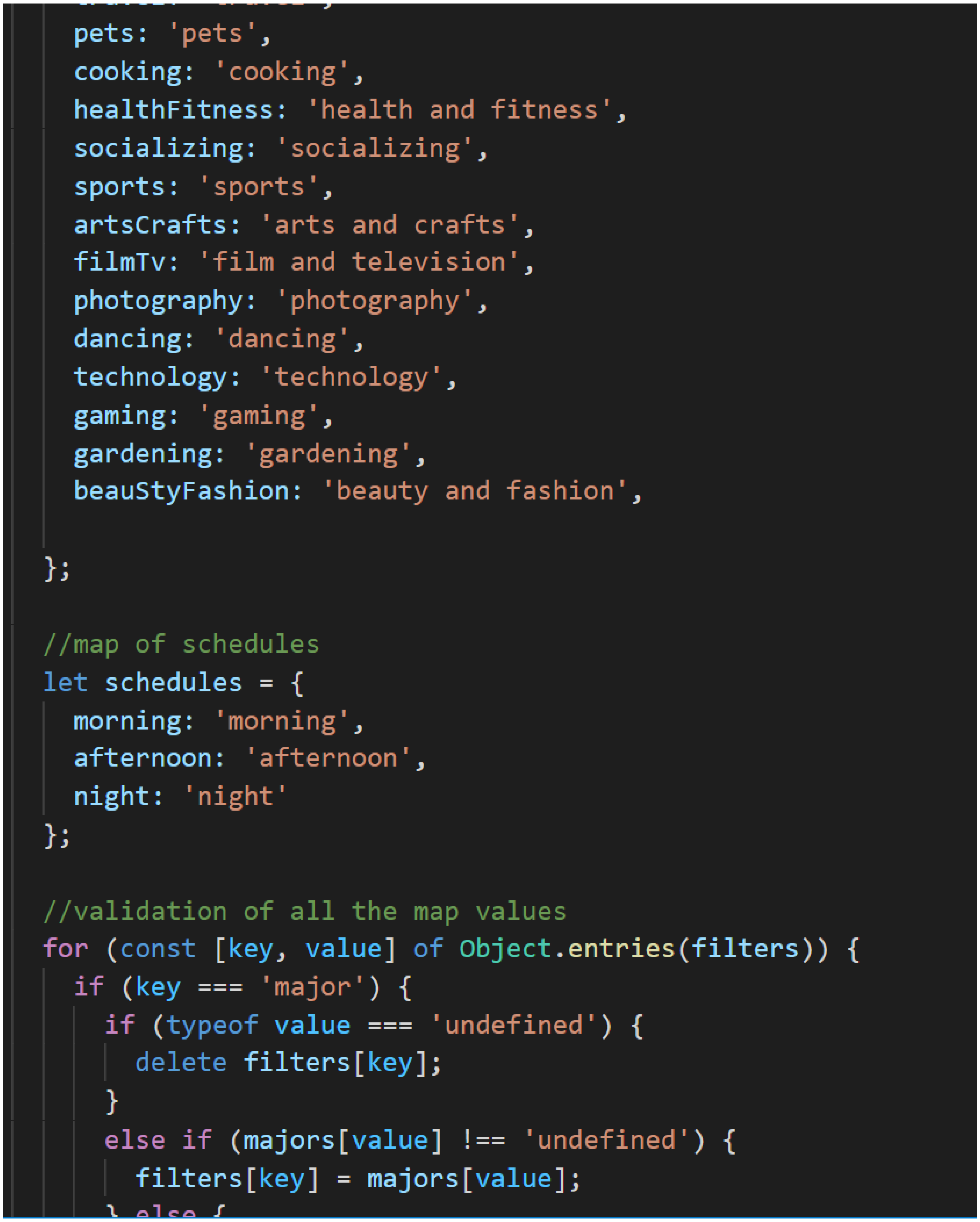
```
/***********************************************************************
 * This function queries the database to find all rows in the student
 * that match the given parameters chosen by the user.
 * Returns: an array of students
 ***********************************************************************/

const getRoommatesByFiltering = async function (filters) {
  if (!(Object.values(filters).length === 0)) {
    const sqlPrepend = 'SELECT BIN_TO_UUID(student.id,true) AS studentId, BIN_TO_UUID(user.id,true) AS userId, institution.name AS institution, student.major, student.personality
    const sqlJoin = ' INNER JOIN institution ON institution.id = student.institutionID INNER JOIN user ON user.id = student.userId WHERE '
    const sqlFilters = Object.keys(filters).map(k => `student.${k} = ?`).join(' AND ')
    // Query the database and return students
    const [students, fields] = await db.query(`${sqlPrepend}${sqlJoin}${sqlFilters}`, Object.values(filters));
    return students;
  }

  const sql = `
    SELECT
      BIN_TO_UUID(student.id,true) AS studentId,
      BIN_TO_UUID(user.id, true) AS userId,
      institution.name AS institution,
      student.major,
      student.personality,
      student.hobby1,
      student.hobby2,
      student.schedule,
      user.username,
      user.emailAddress,
      user.name,
      user.birthdate,
      user.lastSeen,
      user.photo,
      user.verified,
      user.gender
    FROM student
    INNER JOIN user ON user.id = student.userId
    INNER JOIN institution on institution.id = student.institutionID
  `;
  const [students, fields] = await db.query(sql);
  return students
}
```



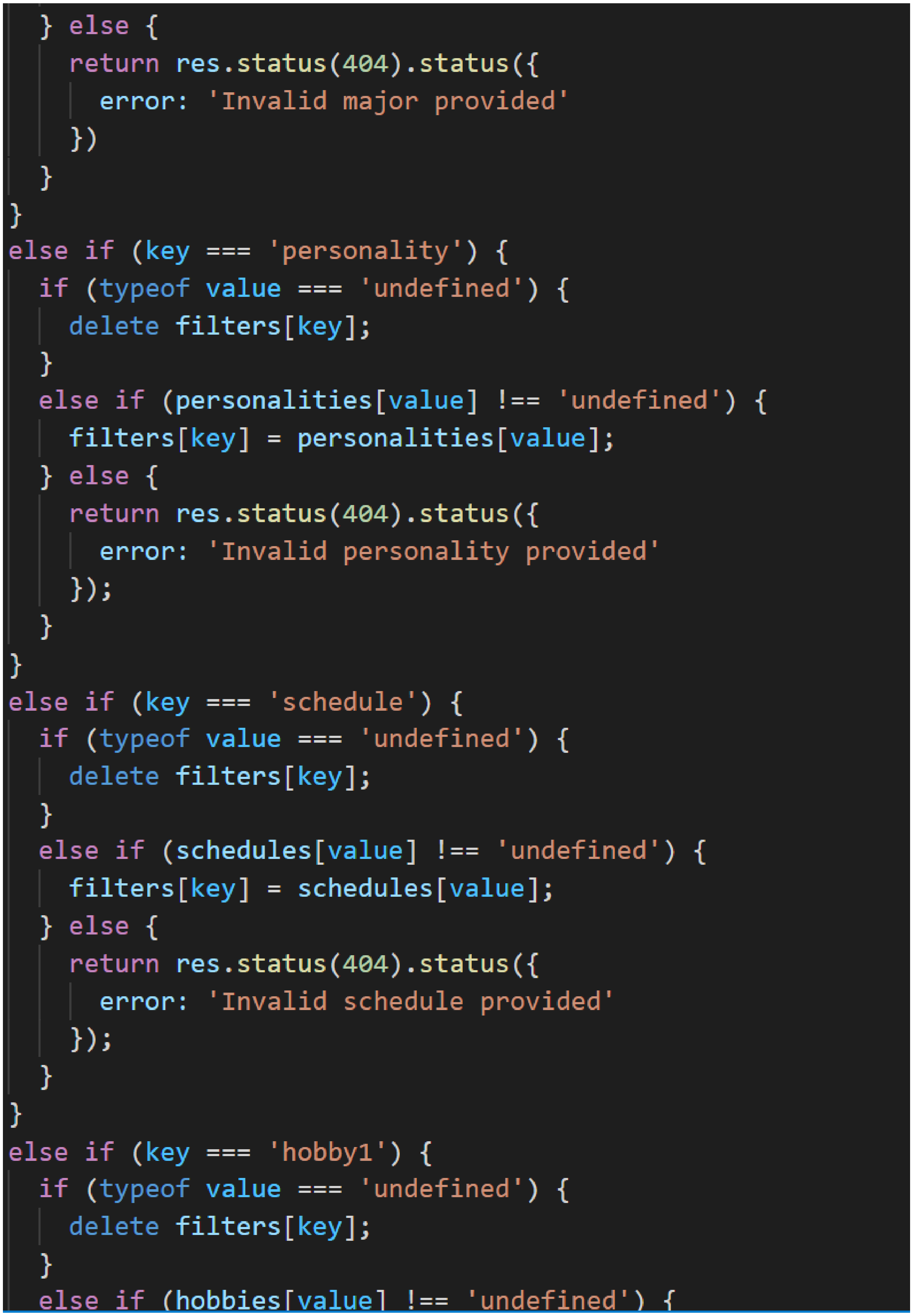
```
/**************************************************************************
 * This function is designed to be used to filter a search for
 * student users by using query parameters. It searches the database
 * any students that have the same parameters that match their attributes
 * Returns:
 * 200 if the request is successful
 * 400 if the request is unsuccessful
 *************************************************************************/


const getRoommateByFiltering = async function (req, res, next) {
  //map of roommate filter options with a validation check to see if the values are not null
  let filters = {
    major: req.query.major,
    personality: req.query.personality,
    hobby1: req.query.hobby1,
    hobby2: req.query.hobby2,
    schedule: req.query.schedule
  };

  //map of majors
  let majors = {
    computerScience: 'computer science',
    physic: 'physics',
    mathematic: 'mathematic',
    biology: 'biology',
    businessManagement: 'business management',
    business: 'business',
    accounting: 'accounting',
    nursing: 'nursing',
    psychology: 'psychology',
    communication: 'communication',
    marketing: 'marketing',
    generalEducation: 'general education',
    elementaryEducation: 'elementary education',
    finance: 'finance',
    criminalJustice: 'criminal justice',
```



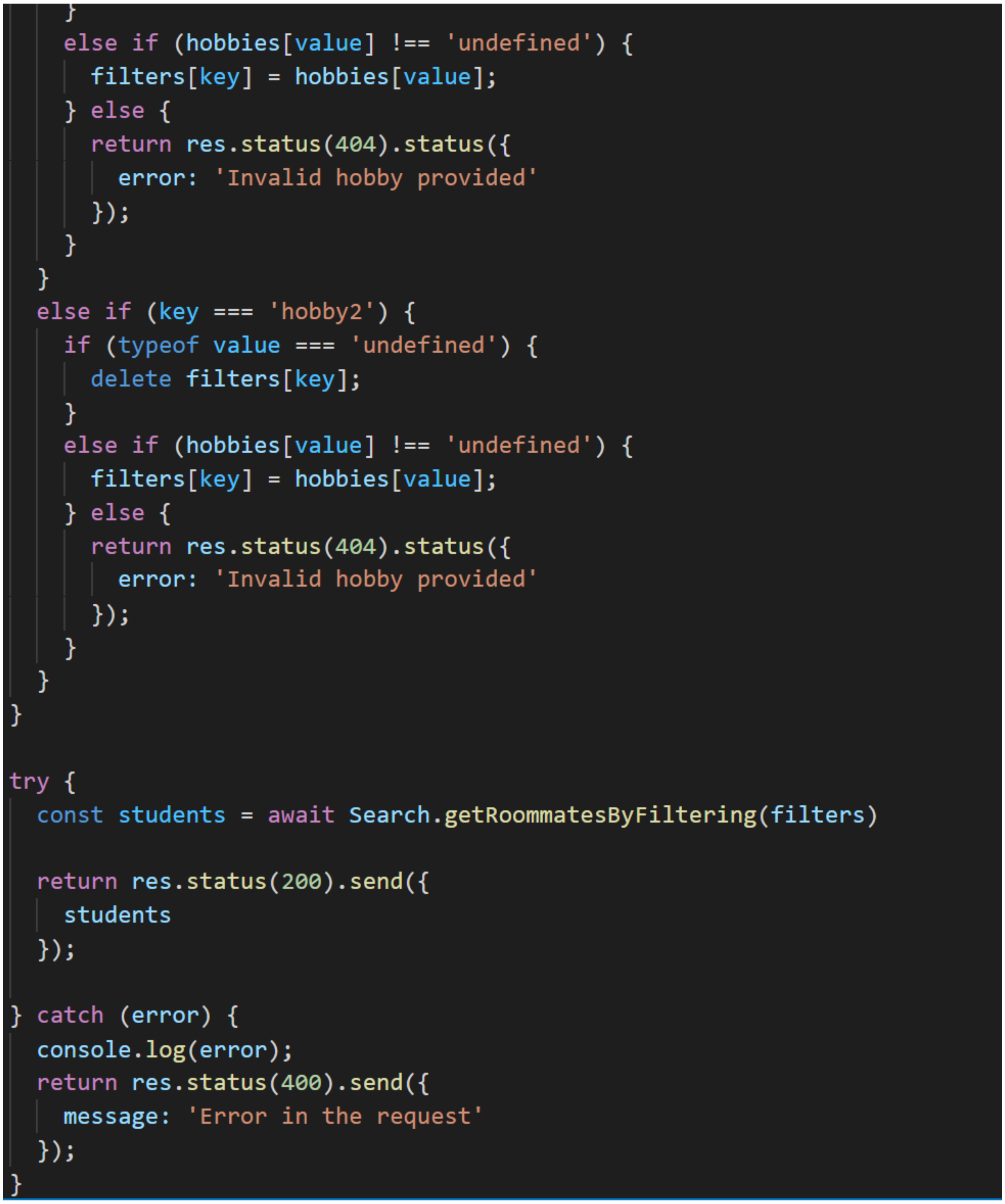
```
  criminalJustice: 'criminal justice',
  politicalScience: 'political science',
  economics: 'economics',
  electricalEngineering: 'electrical engineering',
  history: 'history',
  liberalArts: 'liberal arts',
  sociology: 'sociology'
};

//map of personalities
let personalities = {
  architect: 'architect',
  logician: 'logician',
  commander: 'commander',
  debater: 'debater',
  advocate: 'advocate',
  mediator: 'mediator',
  protagonist: 'protagonist',
  campaigner: 'campaigner',
  logistician: 'logistician',
  defender: 'defender',
  executive: 'executive',
  consul: 'consul',
  virtuoso: 'virtuoso',
  adventurer: 'adventurer',
  entrepreneur: 'entrepreneur',
  entertainer: 'entertainer'
};

//map of hobbies
let hobbies = {
  music: 'music',
  food: 'food',
  readingWriting: 'reading/writing',
  travel: 'travel',
```



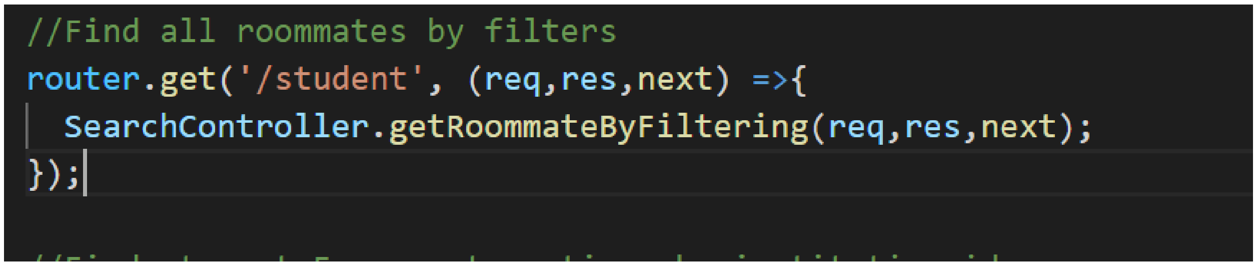
```
  pets: 'pets',
  cooking: 'cooking',
  healthFitness: 'health and fitness',
  socializing: 'socializing',
  sports: 'sports',
  artsCrafts: 'arts and crafts',
  filmTv: 'film and television',
  photography: 'photography',
  dancing: 'dancing',
  technology: 'technology',
  gaming: 'gaming',
  gardening: 'gardening',
  beauStyFashion: 'beauty and fashion',

};

//map of schedules
let schedules = {
  morning: 'morning',
  afternoon: 'afternoon',
  night: 'night'
};

//validation of all the map values
for (const [key, value] of Object.entries(filters)) {
  if (key === 'major') {
    if (typeof value === 'undefined') {
      delete filters[key];
    }
    else if (majors[value] !== 'undefined') {
      filters[key] = majors[value];
    } else {
```



```
  } else {
    return res.status(404).status({
      error: 'Invalid major provided'
    })
  }
}
else if (key === 'personality') {
  if (typeof value === 'undefined') {
    delete filters[key];
  }
  else if (personalities[value] !== 'undefined') {
    filters[key] = personalities[value];
  } else {
    return res.status(404).status({
      error: 'Invalid personality provided'
    });
  }
}
else if (key === 'schedule') {
  if (typeof value === 'undefined') {
    delete filters[key];
  }
  else if (schedules[value] !== 'undefined') {
    filters[key] = schedules[value];
  } else {
    return res.status(404).status({
      error: 'Invalid schedule provided'
    });
  }
}
else if (key === 'hobby1') {
  if (typeof value === 'undefined') {
    delete filters[key];
  }
  else if (hobbies[value] !== 'undefined') {
```



```
    }
    else if (hobbies[value] !== 'undefined') {
      filters[key] = hobbies[value];
    } else {
      return res.status(404).status({
        error: 'Invalid hobby provided'
      });
    }
  }
  else if (key === 'hobby2') {
    if (typeof value === 'undefined') {
      delete filters[key];
    }
    else if (hobbies[value] !== 'undefined') {
      filters[key] = hobbies[value];
    } else {
      return res.status(404).status({
        error: 'Invalid hobby provided'
      });
    }
  }
}

try {
  const students = await Search.getRoommatesByFiltering(filters)

  return res.status(200).send({
    students
  });

} catch (error) {
  console.log(error);
  return res.status(400).send({
    message: 'Error in the request'
  });
}
```



```
//Find all roommates by filters
router.get('/student', (req,res,next) =>{
  SearchController.getRoommateByFiltering(req,res,next);
});
```

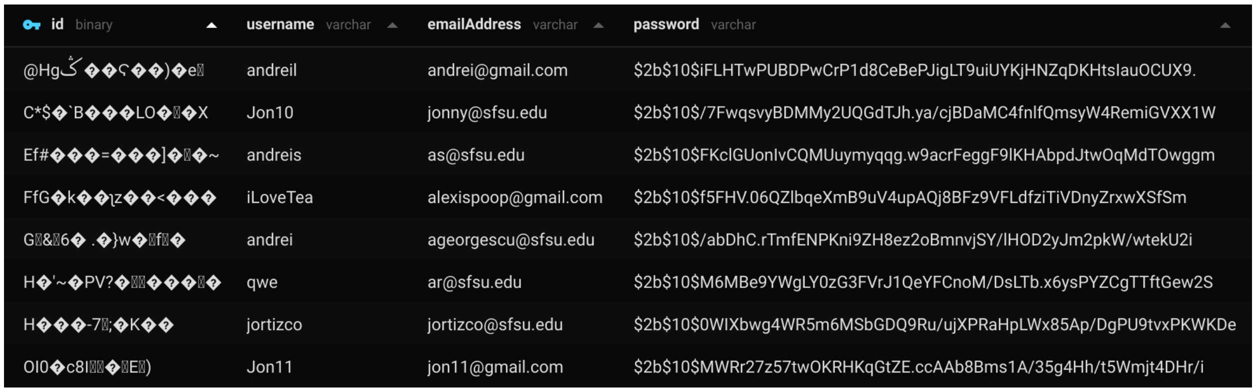
Self-check: Best Practices for Security

Major Assets

- Sensitive User Data (emails & attending university)
- Passwords
- Listing IDs
- Landlord IDs
- Student IDs
- User IDs

Confirmation that PW in the DB are encrypted

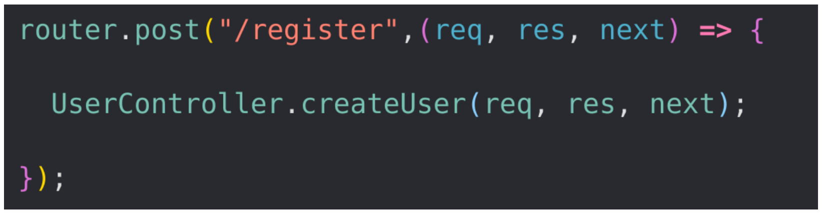
First and foremost, selecting all user rows from our database confirms that passwords are encrypted.



id binary	username varchar	emailAddress varchar	password varchar
@Hgڴ ��ʕ��)�e▯	andreil	andrei@gmail.com	$2b$10$iFLHTwPUBDPwCrP1d8CeBePJigLT9uiUYKjHNZqDKHtsIauOCUX9.
C*$�`B���LO�▯�X	Jon10	jonny@sfsu.edu	$2b$10$/7FwqsvyBDMMy2UQGdTJh.ya/cjBDaMC4fnlfQmsyW4RemiGVXX1W
Ef#���=���]�▯�~	andreis	as@sfsu.edu	$2b$10$FKclGUonIvCQMUuymyqqg.w9acrFeggF9lKHAbpdJtwOqMdTOwggm
FfG�k��ɿz��<���	iLoveTea	alexispoop@gmail.com	$2b$10$f5FHV.06QZlbqeXmB9uV4upAQj8BFz9VFLdfziTiVDnyZrxwXSfSm
G▯&▯6� .�}w�▯f▯�	andrei	ageorgescu@sfsu.edu	$2b$10$/abDhC.rTmfENPKni9ZH8ez2oBmnvjSY/lHOD2yJm2pkW/wtekU2i
H�'~�PV?�▯▯����▯�	qwe	ar@sfsu.edu	$2b$10$M6MBe9YWgLY0zG3FVrJ1QeYFCnoM/DsLTb.x6ysPYZCgTTftGew2S
H���-7▯;�K��	jortizco	jortizco@sfsu.edu	$2b$10$0WIXbwg4WR5m6MSbGDQ9Ru/ujXPRaHpLWx85Ap/DgPU9tvxPKWKDe
OI0�c8I▯▯�▯E▯)	Jon11	jon11@gmail.com	$2b$10$MWRr27z57twOKRHKqGtZE.ccAAb8Bms1A/35g4Hh/t5Wmjt4DHr/i

As for the process; at a high-level it consists of the password being sent to the Auth API and it being hashed before our User model inserts it into the database.

This process all starts with the '/auth/register' route which is a POST route. This route simply calls the 'createUser' method from the User Controller.



```
router.post("/register",(req, res, next) => {

  UserController.createUser(req, res, next);

});
```

The `createUser` method first calls a method that validates the request's body and then if that succeeds it calls another method that interacts with the User model.

```
// ---------------------------------------------------------------
// This function takes in a request body and creates a new user.
// Returns:
//      200 if successful, 400 if something went wrong
//      JSON with a message field
// ---------------------------------------------------------------
const createUser = async function (req, res, next) {
  // Validating registration form
  const bodyValidated = await validateRegistrationBody(req, res, next);
  if(bodyValidated === true) {
    await registerNewUser(req, res, next);
  }
};
```

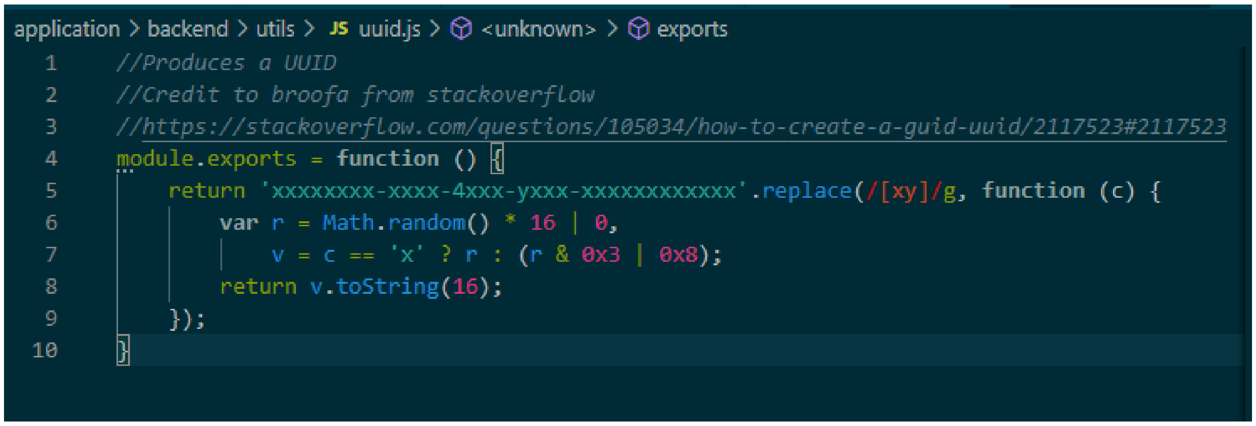
Inside of the `registerNewUser` method is where a user's password gets encrypted. Specifically we use **Bcrypt** which is a fairly secure password hashing algorithm.

```
  try {
    // Hashing the password and creating the user
    const hashedPassword = await hashPassword(password);
    const userCreated = await User.createNewUser(
      username,
      email,
      hashedPassword,
      name,
      dob,
      gender,
    );

    if (userCreated) {
      return res.status(200).send({
        error: null,
        message: 'User Created',
        id: userCreated,
      });
    }

    // If the user was not created, return an error
    return res.status(200).send({
      error: 'Could not create a new user.',
      message: 'User not created',
    });

  } catch (e) {
    console.log(e);
    return res.status(200).send({
      error: 'Could not create a new user.',
      message: 'User not created',
    });
  }
}
```

It's important to note here that we placed the `hashPassword` method inside of a try-catch block. This means that if anything goes wrong during the hashing process, the user's data will never get inserted into the database and therefore we avoid accidentally storing unhashed passwords.

Confirmation of User, Student, Landlord and Listing IDs

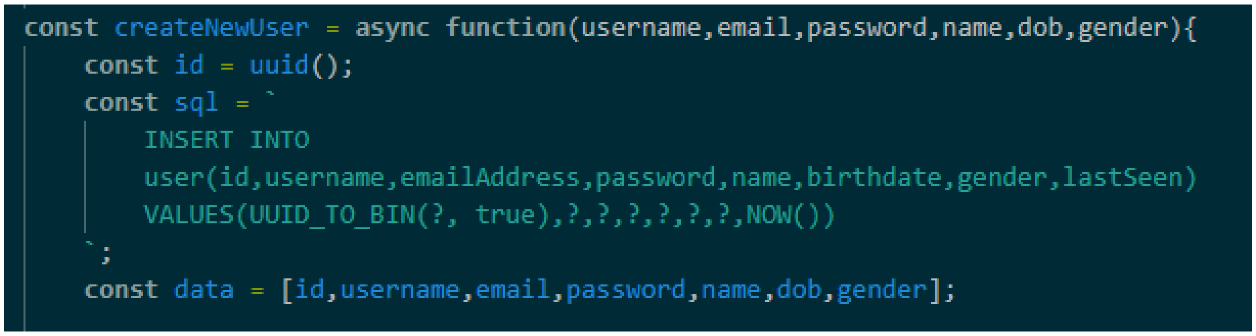
For the IDs of each entity, we used a function to generate a UUID (Universally Unique ID)to create a binary key. This key adds extra security by generating a 128-bit number to identify each entity.



application > backend > utils > JS uuid.js > <unknown> > exports

```
//Produces a UUID
//Credit to broofa from stackoverflow
//https://stackoverflow.com/questions/105034/how-to-create-a-guid-uuid/2117523#2117523
module.exports = function () {
    return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function (c) {
        var r = Math.random() * 16 | 0,
            v = c == 'x' ? r : (r & 0x3 | 0x8);
        return v.toString(16);
    });
}
```

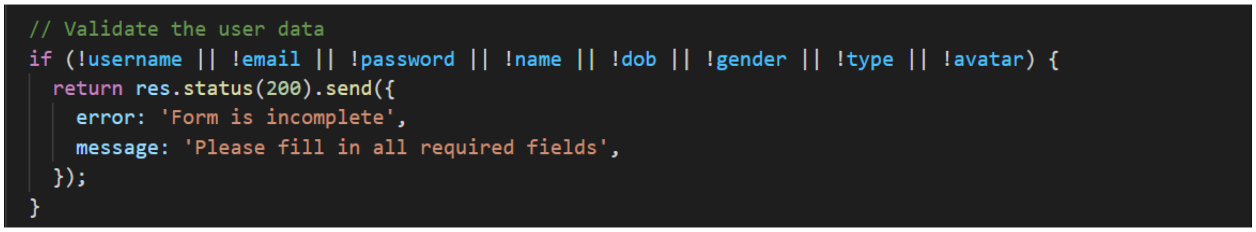
Everytime we create one of these new entities, we use the UUID() function to generate this key and store it into the DB.



```
const createNewUser = async function(username,email,password,name,dob,gender){
    const id = uuid();
    const sql = `
        INSERT INTO
        user(id,username,emailAddress,password,name,birthdate,gender,lastSeen)
        VALUES(UUID_TO_BIN(?, true),?,?,?,?,?,?,NOW())
    `;
    const data = [id,username,email,password,name,dob,gender];
```

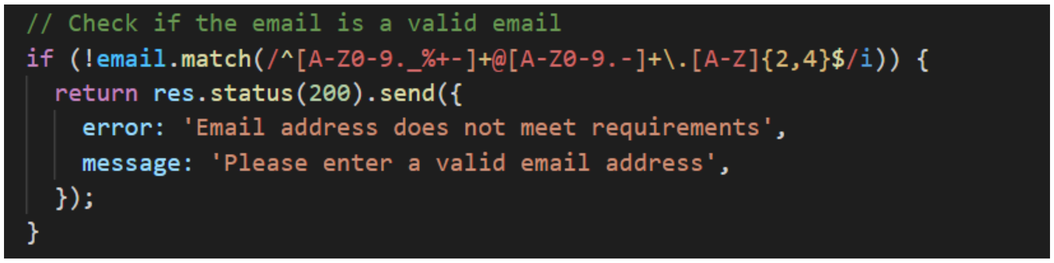
Confirming Input Data Validation on the Backend

In the backend we perform validation checks for when a user registers for an account on our service.



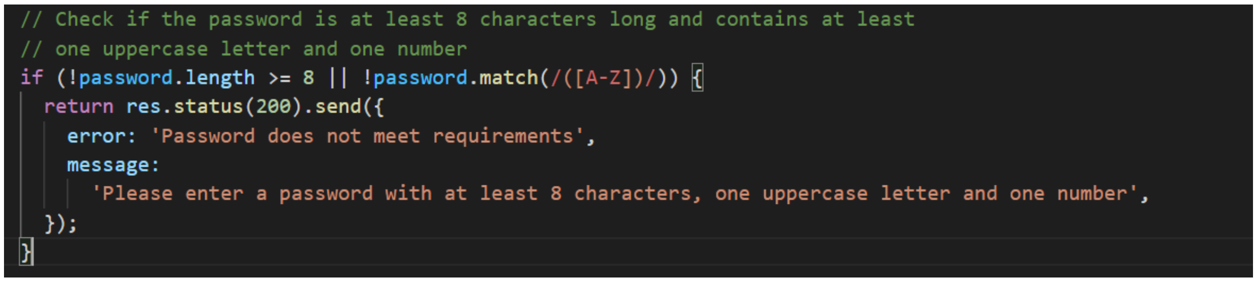
```
// Validate the user data
if (!username || !email || !password || !name || !dob || !gender || !type || !avatar) {
  return res.status(200).send({
    error: 'Form is incomplete',
    message: 'Please fill in all required fields',
  });
}
```

In the code above we check if all values within the registration form are fully completed and ask the user to fill in all fields of the form if they are not.



```javascript
// Check if the email is a valid email
if (!email.match(/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/i)) {
  return res.status(200).send({
    error: 'Email address does not meet requirements',
    message: 'Please enter a valid email address',
  });
}
```

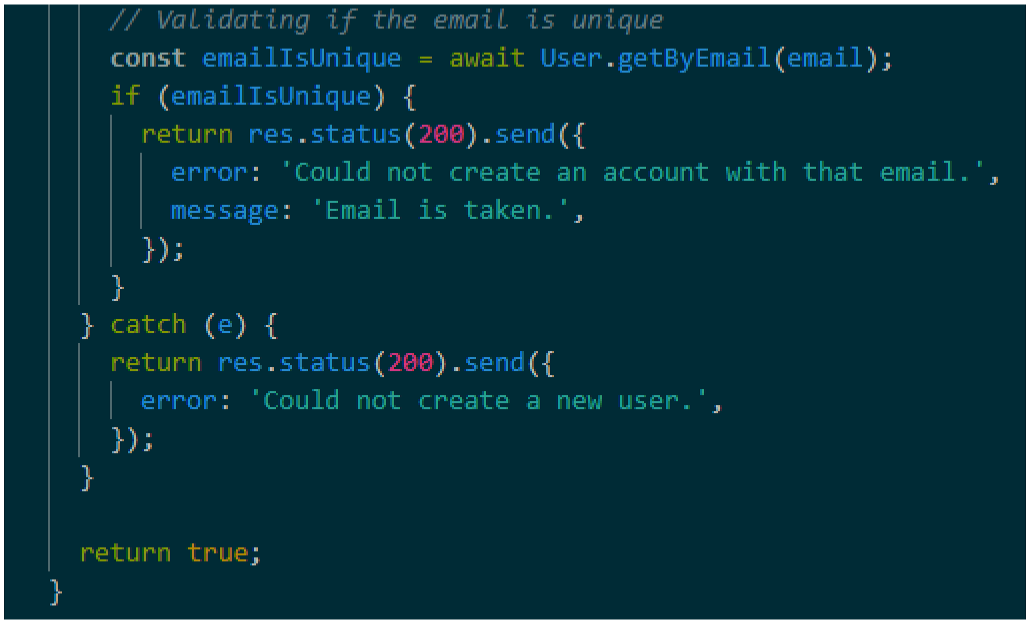
Within the code above we check whether the email they input is of a valid format and if it fails then we ask them to re-enter a valid format.



```javascript
// Check if the password is at least 8 characters long and contains at least
// one uppercase letter and one number
if (!password.length >= 8 || !password.match(/([A-Z])/)) {
  return res.status(200).send({
    error: 'Password does not meet requirements',
    message:
      'Please enter a password with at least 8 characters, one uppercase letter and one number',
  });
}
```

We check if the user enters in a password that meets a format of a minimum of 8 characters lengths and 1 uppercase letter with 1 number in the password. If it fails we display a message stating as such and have the user enter in a valid password.

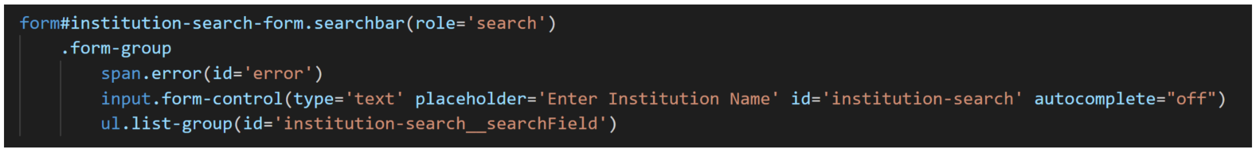
We also created helper functions that verify the uniqueness of a Username, Email. This is done by comparing the input with the database to ensure no duplicate emails or usernames.



```javascript
      // Validating if the email is unique
      const emailIsUnique = await User.getByEmail(email);
      if (emailIsUnique) {
        return res.status(200).send({
          error: 'Could not create an account with that email.',
          message: 'Email is taken.',
        });
      }
    } catch (e) {
      return res.status(200).send({
        error: 'Could not create a new user.',
      });
    }

    return true;
  }
```

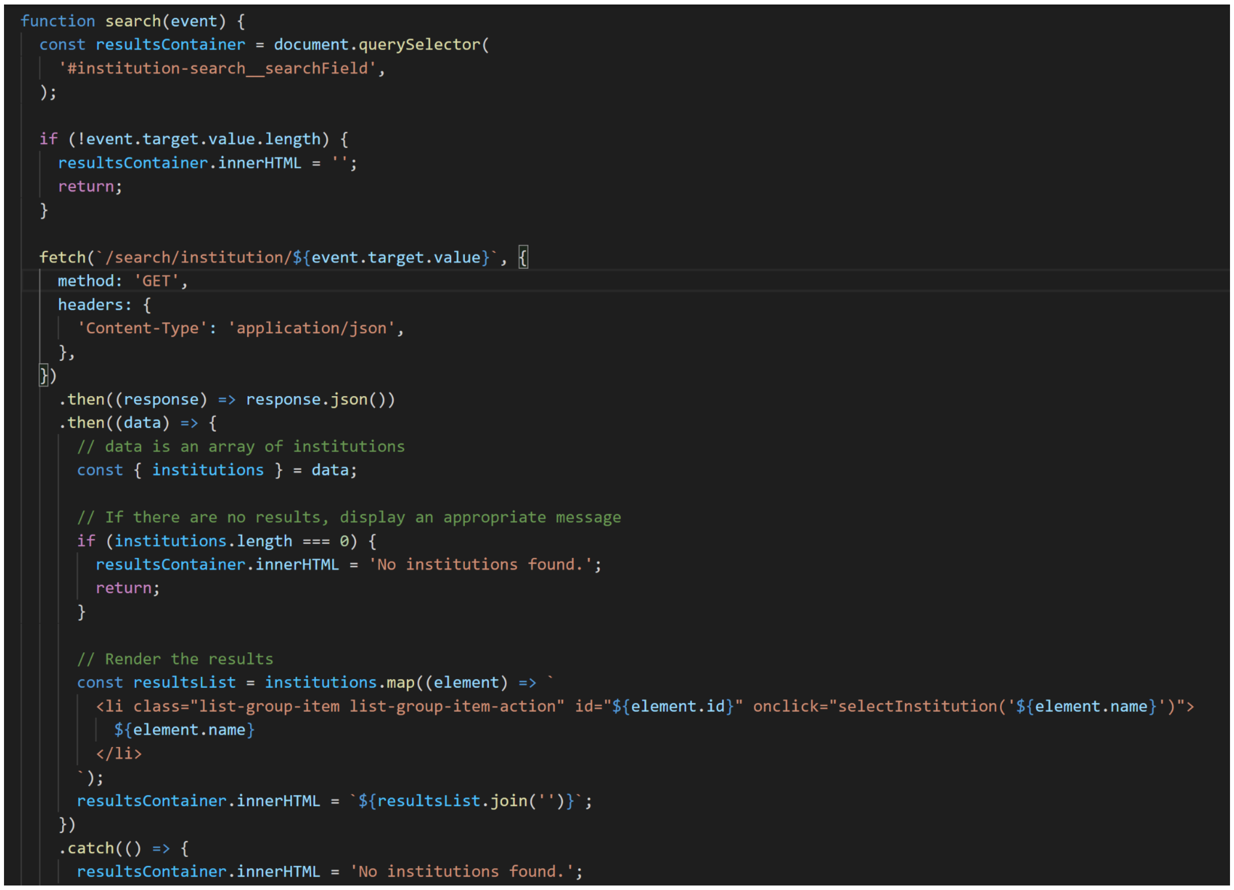
Confirming Input Data Validation on the Frontend

In the frontend, the search bar input is created on the home pug view using form.



```
form#institution-search-form.searchbar(role='search')
    .form-group
        span.error(id='error')
        input.form-control(type='text' placeholder='Enter Institution Name' id='institution-search' autocomplete="off")
        ul.list-group(id='institution-search__searchField')
```

The search bar is used to search for a user's desired institution and was implemented in the following javascript function.



```
function search(event) {
  const resultsContainer = document.querySelector(
    '#institution-search__searchField',
  );

  if (!event.target.value.length) {
    resultsContainer.innerHTML = '';
    return;
  }

  fetch(`/search/institution/${event.target.value}`, {
    method: 'GET',
    headers: {
      'Content-Type': 'application/json',
    },
  })
    .then((response) => response.json())
    .then((data) => {
      // data is an array of institutions
      const { institutions } = data;

      // If there are no results, display an appropriate message
      if (institutions.length === 0) {
        resultsContainer.innerHTML = 'No institutions found.';
        return;
      }

      // Render the results
      const resultsList = institutions.map((element) => `
        <li class="list-group-item list-group-item-action" id="${element.id}" onclick="selectInstitution('${element.name}')">
          ${element.name}
        </li>
      `);
      resultsContainer.innerHTML = `${resultsList.join('')}`;
    })
    .catch(() => {
      resultsContainer.innerHTML = 'No institutions found.';
```

The backend has a list of institutions which is accessed in the frontend by using fetch. Once the user begins to fill the input, the code will then check the value's length to see if the data is available in the backend. Based on the user's inputs, it will display the data results and no results if none.

Self-check: Adherence to Original Non-functional Specs

Functionality

Requirement	**Status**
The website should utilize all tools and frameworks approved by the CTO.	DONE
The website should be easy to use and intuitive.	DONE
The website should have a simple and non-cluttered interface.	DONE
The website should be responsive across all modern devices.	DONE
The website will use Amazon Web Services for deployment.	DONE
The website will use Amazon Web Services for its database.	DONE
The website should use HTTPS for all requests.	DONE

Security

Requirement	**Status**
Users must authenticate themselves before accessing any protected pages.	DONE
Users must authenticate themselves if their cookie is expired.	DONE
The student dashboard page must only be available to verified student users.	DONE
The landlord dashboard page must only be available to landlord users.	DONE
Registered users should be able to view their own chat messages.	ON TRACK
Registered users should be able to send messages only within their group.	ON TRACK
All sensitive information must be encrypted before stored in the database.	DONE

Privacy

Requirement	Status
Only registered users will be able to view all listings.	DONE
Only registered users will be able to view students.	DONE
Landlords will not have access to viewing other landlords.	DONE
Landlords will not be able to search student data.	DONE
Landlords will not be able to search landlords.	DONE
Registered users’ chat messages should remain private.	ON TRACK

Legal

Requirement	Status
All users must accept the terms and service policy before creating an account.	DONE
All users must accept the privacy policy before creating an account.	DONE
All landlords must prove ownership of a listing.	ON TRACK
The website must have a copyright notice.	DONE
The website must have a privacy policy notice.	DONE
The website must have a terms and conditions notice.	DONE
The website must have a cookie notice.	ON TRACK
All content uploaded to the site must be owned by the user who is uploading it.	DONE

Performance

Requirement	Status
The frontend must have processes in place that prevent it from being offline.	DONE
The backend must have processes in place that prevent it from being offline.	DONE
The website load time should be within industry standard requirements.	DONE

System Requirements

Requirement	Status
The website shall work up to version 91.0.4472.106 of Google Chrome.	DONE
The website shall work up to version _ of Safari.	DONE
The website shall work up to version _ of Microsoft Edge.	DONE
The website shall work up to version _ of Mozilla Firefox.	DONE
The website shall work up to version _ of Android.	DONE
The website shall work up to version _ of iOS.	DONE
The website will be supported in the English language.	DONE

Marketing

Requirement	Status
The website should follow SEO best practices.	DONE
Each page on the website shall have the logo on the navigation bar.	DONE
Each page will be clear and easy to navigate for new visitors.	DONE
Each user shall be able to connect their account with their social media platforms.	ON TRACK

Content

Requirement	Status
The website will have a navigation bar.	DONE
The website navigation bar will direct users to different pages.	DONE
The website pages will have a footer.	DONE
The website will have a scalable map.	DONE
The website should give registered users the option to private message.	ON TRACK

Scalability

Requirement	Status
The website should be capable of handling a large number of listings.	DONE
The website should be composed of a frontend and backend which are separate codebases.	DONE
The website shall be able to handle a large number of users.	DONE
The chat rooms shall be able to handle a large number of users.	DONE

Capability

Requirement	Status
The website should process all requests as expected by the users.	DONE
The website should respond with a descriptive error if one occurs.	DONE
The website should alert users when they are about to leave the site.	DONE

Look and Feel

Requirement	Status
The navigation bar should have a logo.	DONE
The navigation bar should have a dark colored background.	DONE
The navigation bar should have a light shade hover color button.	DONE
The footer should have a logo.	DONE
The footer should have a sitemap with all site pages.	DONE
The website should have a plain color layout.	DONE
The website should have a simple layout.	DONE
The website will have a readable font.	DONE
The website elements fonts will be uniform.	DONE
The website elements will be continuous.	DONE
The website’s pages will be scrollable in the vertical axis.	DONE
The font should be roman new times.	DONE

The feeling should be friendly.	DONE
The website should not be repetitive.	DONE
The website should be easy to traverse.	DONE
Pages should be instant loaded.	DONE
The private account page should be easy to find.	DONE
The private chat font will be easy to read and uniform.	DONE
The private chat should be easily identifiable.	DONE
The map should be easily identifiable.	DONE
The map key will be easily identifiable.	DONE
Profiles will clearly display a user’s role.	DONE
The post should be easily identifiable.	DONE
The forum should be easily identifiable.	DONE
The buttons should be easily identifiable.	DONE
Listing filters should be easily identifiable.	DONE

Coding Standards

Requirement	**Status**
All code must be reviewed before it is merged with any of the three main branches.	DONE
All code must be submitted via pull requests.	DONE
All code must be pushed to proper branches.	DONE
All code must be documented.	DONE
All code should be organized.	DONE
There should be no repetitive code.	DONE
There should be no unused code.	DONE
All code should have in-line comments where needed.	DONE
The code should have a uniform formatting style.	DONE
The backend code should use an object-oriented programming paradigm.	DONE
The backend must implement methods to prevent SQL injection.	DONE

Availability

Requirement	Status
The frontend must be online at all times.	DONE
The backend must be online at all times.	DONE
The website is updated if and only if code is pushed to the master branch.	DONE
The website will resync if a loss of connection occurs.	DONE
The website shall display error messages when errors occur.	DONE
The website will be managed on a PST timezone.	DONE

Cost

Requirement	Status
Amazon web services server is free.	DONE
Amazon web services relational database is free.	DONE
Server must not exceed the free tier.	DONE
Server maintenance is free.	DONE

Storage

Requirement	Status
Store users profile data on the database.	DONE
Store landlords listings on the database.	DONE
Remove listings from the database after it has been deleted by the user.	DONE
Store up to 60 days of inactive listings (incase user wants to repost).	DONE
Remove listings from the database after 60 days of inactivity.	ON TRACK
Repost will restart the 60-day clock of storage time.	ON TRACK
Store students' chat history on the database.	ON TRACK
Store usernames on the database	DONE
Store emails on the database	DONE
Store passwords on the database	DONE

Store landlord information on the database.	DONE
Store landlord photos on the database.	DONE
Store student photos on the database.	DONE
Store error logs on the database.	ON TRACK

Expected Load

Requirement	**Status**
The website will be able to handle as many users as AWS can support.	DONE
The website will be able to handle as many listings as AWS can support.	DONE

Detailed List of Contributions

Andrei

- Hosted 1x team meeting where we worked on the product summary, discussed priority 1 functional requirements, and other milestone 4 related tasks.
- Hosted pair programming sessions with the frontend and the backend teams.
- Edited and formatted the milestone 4 document.
- Worked on connecting the frontend to the backend api.

Jonathan

- Hosted meeting with the backend team to discuss Usability test plan.
- Hosted multiple pair coding sessions to get back end code working.
- Worked on the Usability test plan.
- Worked on the QA Test Plan.
- Worked with Alexandre on Code review.
- Worked on the P1 compromise
- Worked on Self Check Best Practices for Security

Meeka

- Hosted meeting with the frontend team to discuss frontend feedback from the entire team.
- Hosted meeting with the frontend team to discuss QA test plan.
- Worked on the QA Test Plan
- Worked on Self Check Best Practices for Security
- Worked on Unique features for Product Summary
- Implemented feedback given by the team on the FAQ, listing, profile, and search pages.

Jimmy

- Implemented feedback given by the team on the dashboard page.
- Implemented Roommates Near you
- Participated in pair programming

Alexandre

- Worked on the Usability test plan.
- Worked on Code review with Jonathan.
- Participated in coding sessions to get back end code working.

Sayed

- Implemented FrontEnd Feedback to get buttons working on Safari

Ahad

- Implementing Chat API on the backend